WHAT IS CLAIMED IS:

1. A computer readable medium encoding of a data structure implementation, the encoding comprising:

a definition of a double-ended array instantiable in memory; and

a functional encoding of opposing-end access operations that, when executed on respective one or more processors that access the memory, provide concurrent push-type and pop-type access at at least one of the opposing ends and concurrent, opposing-end accesses are non-interfering for at least some states of the array,

wherein the data structure implementation is linearizable and non-blocking, and

wherein concurrent execution of the access operations is mediated using a single-target synchronization primitive.

2. The data structure encoding of claim 1,

wherein the concurrent opposing-end access operations are non-interfering for all but boundary condition states of the array.

3. The data structure encoding of claim 1,

wherein the non-blocking implementation is obstruction-free, though not wait-free or lock-free.

4. The data structure encoding of claim 1,

wherein the single-target synchronization primitive employs a Compare-And-Swap (CAS) operation.

5. The data structure encoding of claim 1,

wherein the single-target synchronization primitive employs a Load-Linked (LL) and Store-Conditional (SC) operation pair.

6. The data structure encoding of claim 4,

wherein the single-target of the single-target synchronization primitive includes a value encoding for an element of the array and a version number encoded integrally therewith.

7. The data structure encoding of claim 1,
wherein the double-ended array implements a deque.

8. The data structure encoding of claim 1,
wherein the opposing-end access operations are at least consistent with semantics of a FIFO queue.

9. The data structure encoding of claim 1,
wherein the boundary-condition states include an empty state.

10. The data structure encoding of claim 1,
wherein the boundary-condition states include a single element state.

11. The data structure encoding of claim 1,
wherein the array is indexable as a circular array.

12. The data structure encoding of claim 11,
wherein the boundary-condition states include a full state.

13. The data structure encoding of claim 11,
wherein the opposing-end accesses include opposing-end, push-type accesses; and
wherein the boundary-condition states include a nearly full state.

14. The data structure encoding of claim 1,
wherein distinct left null and right null distinguishing values are employed to identify free elements of the array.

15. The data structure encoding of claim 14,
wherein the array is indexed as a circular array; and

wherein an additional distinguishing value is employed to facilitate consumption of free elements by push-type operations at either end of the array.

16. The data structure encoding of claim 1,
embodied as a software component combinable with program code to provide the program code with non-blocking access to a concurrent shared object.

17. The data structure encoding of claim 1,
embodied as a program executable to provide non-blocking access to a concurrent shared object.

18. The data structure encoding of claim 1,
wherein the computer readable medium includes at least one medium selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium and a network, wireline, wireless or other communications medium.

19. A single-target synchronization primitive based, non-blocking, fully funcational deque implementation for which concurrent opposing-end access operations do not always interfere, and wherein shared storage usage is insensitive to a number of access operations that concurrently access the deque.

20. The CAS-based non-blocking deque implementation of claim 19,
wherein the implementation is obstruction-free, though not wait-free or lock-free.

21. The non-blocking deque implementation of claim 19,
wherein the concurrent opposing-end access operations are non-interfering for all but boundary condition states.

22. The non-blocking deque implementation of claim 19,
wherein state of the deque is encoded using an array.

23. The non-blocking deque implementation of claim 22,
wherein the array is a circular array.

24. The non-blocking deque implementation of claim 19,
wherein the single-target synchronization includes use of a Compare-And-Swap (CAS) operation.

25. The non-blocking deque implementation of claim 19,
wherein the single-target synchronization includes use of a Load-Linked (LL) and Store-Conditional (SC) operation pair.

26. The non-blocking deque implementation of claim 19,
wherein at least some concurrently executed access operations interfere with each other; and
wherein the interfering concurrently executed access operations are each retried.

27. The non-blocking deque implementation of claim 26,
wherein the non-blocking deque implementation does not guarantee that at least one of the interfering concurrently executed access operations makes progress.

28. The non-blocking deque implementation of claim 27,
wherein a separate contention management facility is employed to ensure progress in a concurrent computation that employs the deque implementation.

29. The non-blocking deque implementation of claim 19,
embodied as a software that defines a representation of the deque instantiable in memory and which includes a functional encoding of access operations executable by one or more processors to operate on state of the deque.

30. A method of managing obstruction-free access to a shared double-ended array, the method comprising:

instantiating the double-ended array in memory; and

operating on state of the array using access operations that detect interference by other executions thereof using a single-target synchronization primitive; and

after detection of an interfering execution, retrying an interfered-with access operation,

wherein execution of respective ones the access operations allows at least (i) concurrent push-type and pop-type access at at least one of the opposing ends and (ii) concurrent, opposing-end accesses that are non-interfering for at least some states of the array.

31. The method of claim 30,

wherein the concurrent, opposing-end accesses are non-interfering for all but boundary-condition states of the array.

32. The method of claim 30,

wherein execution of the access operations is obstruction-free, though not wait-free or lock-free.

33. The method of claim 30,

wherein the single-target synchronization primitive employs a Compare-And-Swap (CAS) operation.

34. The method of claim 30,

wherein the single-target synchronization primitive employs a Load-Linked (LL) and Store-Conditional (SC) operation pair.

35. The method of claim 30,

wherein the double-ended array includes a representation of a deque; and

wherein the access operations include both push-type and pop-type access operations at both opposing ends of the deque.

36. The method of claim 30,
wherein a contention management facility facilitates progress of access operations.

37. The method of claim 36, further comprising:
changing, during the course of a computation involving the shared double-ended array, a contention management strategy employed by the contention management facility.

38. The method of claim 36, further comprising:
operating the separate contention management facility.

39. The method of claim 30,
wherein progress is ensured not by the shared object implementation, but rather by a separate contention management facility.

40. An apparatus comprising:
one or more processors;
one or more data stores addressable by each of the one or more processors; and
means for coordinating concurrent non-blocking execution, by one or more of the processors, of at least opposing-end push-type and pop-type access operations on a fully functional deque data structure encoded in the one or more data stores, the coordinating employing a compare-and-swap (CAS) synchronization primitive to detect interference of concurrently executed ones of the access operations, the coordinating means ensuring that, for all but boundary-condition states of the deque, opposing-end accesses are non-interfering.

41. The apparatus of claim 40,
wherein the coordinating means tolerates non-progress of interfering executions of the access operations.

42. The apparatus of claim 40, further comprising:

means for managing contention between interfering executions of the access operations.

43. A non-blocking method of operating on a double-ended queue data structure, the method comprising:

concurrently executing push-type and pop-type access operations at at least one of opposing ends of the double-ended queue;

detecting interference with a particular execution of one of the access operations using a single-target synchronization primitive; and

tolerating, in the implementation of the double-ended queue data structure, a possibility that two or more executions of the access operations interfere with each other and each consequently fail to make progress,

wherein the non-blocking property is achieved while ensuring that, for all but boundary-condition states of the deque, opposing-end accesses are non-interfering and without use of a multi-target synchronization primitive.

44. The method of claim 43, further comprising:

managing the possibility that access operations interfere with each other and consequently fail to make progress using a substitutable contention management facility separable from implementation of the double-ended queue data structure.